RFC 5925 : The TCP Authentication Option

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 22 juin 2010

Date de publication du RFC : Juin 2010

https://www.bortzmeyer.org/5925.html

Un des problèmes traditionnels de sécurité de l'Internet est la difficulté à s'assurer que le correspondant est bien celui qu'il prétend être. Même une authentification au niveau de l'application ne suffit pas car rien ne garantit qu'un attaquant ne va pas se glisser dans la communication **après** l'authentification. En effet, rien ne lie la connexion TCP sous-jacente (couche 4) à la communication authentifiée en couche 7. Ce risque de détournement d'une connexion authentifiée est particulièrement important pour les longues sessions comme celles entre deux routeurs BGP ou LDP. Deux méthodes ont été normalisées pour cela, IPsec (RFC 4301 [1]) et TLS (RFC 5246). La première, complexe à déployer entre organisations, n'a pas été un grand succès et la seconde nécessite une adaptation des protocoles applicatifs. Que reste t-il comme solution ?

Il y en a bien d'autres, bien sûr, comme SSH, mais plusieurs protocoles, à commencer par BGP, n'avaient pas envie de s'adapter et cherchaient à sous-traiter le problème de l'authentification et de l'intégrité de la connexion à des couches inférieures, IP ou TCP. La complexité d'IPsec en ayant fait reculer beaucoup, la solution de loin la plus fréquente dans le monde des opérateurs BGP (cf. RFC 4953) était la signature TCP du RFC 2385. Elle fonctionne en calculant un condensat cryptographique du paquet TCP et d'un secret partagé entre les deux machines ("*MD5 password*" dans le vocabulaire BGP). Grâce à l'inclusion de ce secret, un tiers, même s'il a accès au réseau sous-jacent, ne peut pas injecter de paquets TCP prétendant venir du pair.

Cette technique fonctionne mais a de nombreuses limites, comme le fait d'être lié à une fonction de hachage particulière, MD5, que des années de cryptanalyse ont laissé en piteux état <http://www.infosec.sdu.edu.cn/uploadfile/papers/How%20to%20Break%20MD5%20and%20Other%20Hash%20Functions.pdf>. Notre RFC 5925 la remplace donc par une meilleure solution, l'**option TCP d'authentification**.

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc4301.txt

Cette option offre en outre une meilleure protection contre le rejeu, et d'une manière générale une meilleure spécification. La section 1 du RFC rappelle le long historique de cette option. La section 3 résume les raisons de son introduction : la principale est le perfectionnement régulier des attaques contre MD5, alors que le RFC 2385 ne permet pas d'utiliser d'autres algorithmes. En prime, il ne permet pas de changer les clés facilement (aucune fonction de gestion de clés, même très sommaire).

La section 3 présente aussi les principes du nouvel algorithme : fonctionnement identique à celui de son prédécesseur (calcul d'un condensat cryptographique sur le paquet, avec concaténation d'un secret), pas de solution complète de gestion de clés (l'espace pour les options dans l'en-tête TCP est bien trop faible pour cela), mais il fonctionne avec plusieurs algorithmes. D'une manière générale, il suit les recommandations du cahier des charges (jamais publié en RFC) `draft-bellovin-tcpsec` (voir aussi la section 12 pour un bilan détaillé de cette nouvelle option par rapport à son cahier des charges).

Quelles sont les applications possibles de cette option d'authentification (section 3.1) ? Essentiellement les sessions TCP de longue durée comme avec BGP (où les sessions de plusieurs semaines ne sont pas rares). Si tous les exemples actuels d'usage concernent des protocoles de routage, l'option ne leur est pas spécifique (même si le RFC ne le cite pas, on peut penser aussi aux longs flux de certaines applications de communication audio ou vidéo, dont la session de contrôle peut utiliser TCP). Comme IPsec est la solution officielle de l'IETF à tous les problèmes de sécurité, la même section rappelle que cette option ne remplace pas IPsec, même si personne n'en tiendra compte. La section 3.1 note aussi que l'option d'authentification TCP de notre RFC ne protège que la session, pas les données qui y transitent, qui ont pu être relayées par plusieurs acteurs (dans le cas de BGP, par plusieurs routeurs). TCP-AO ("*TCP Authentication Option*") ne remplace donc pas les différents projets de BGP « sécurisé ». Bref, TCP-AO enrichit la palette des outils de sécurisation des protocoles mais ne prétend absolument pas être la solution à tous les problèmes.

La section 3.2 fournit un résumé de la nouvelle option TCP-AO. Elle utilise le mécanisme d'options décrit dans la section 3.1 du RFC 793, avec un nouveau numéro, 29, réservé `<https://www.iana.org/assignments/tcp-parameters/tcp-parameters.xhtml#tcp-parameters-1>` à l'IANA (cf. section 14 du RFC), différent du numéro 19 du RFC 2385.

Contrairement à IPsec, TCP-AO n'a pas de "*Security Index*" (un identificateur unique pour une association de sécurité) explicite mais utilise pour cela le 4-tuple {adresse IP source, port source, adresse IP de destination, port destination}. Il ne permet pas le chiffrement et donc n'assure pas la confidentialité des données, seulement leur authentification et leur intégrité. De plus, il est spécifique à TCP et ne protège donc pas les paquets ICMP (cf. section 9.8).

La section 4 spécifie rigoureusement la nouvelle option numérotée 29. Son format figure en section 4.2 (l'ancien est rappelé en 4.1). La principale nouveauté est le champ "*KeyID*", un octet qui indique la structure de données qui a été utilisée pour générer les clés de session. Cette structure, le MKT ("*Master Key Tuple*") est décrite en détail en section 5.1. Il est important de noter que l'algorithme de hachage choisi n'apparait pas dans l'option TCP-AO (contrairement à des protocoles comme, par exemple, DNSSEC) mais qu'on l'obtient indirectement.

Et la gestion des clés ? Comment se fait-elle avec TCP-AO ? La section 5 explique les deux jeux de clés de TCP-AO : les clés maîtresses (**MKT**, pour "*Master Key Tuple*") et les clés de session ("*traffic keys*"). Les MKT contiennent les paramètres de cryptographie, et elles servent à fabriquer des clés de session. Le MAC dans les paquets TCP est calculé à partir des clés de session.

Les MKT sont présentées en détail en section 5.1. Une MKT comprend :

— L'identificateur du MKT. C'est une paire, l'identificateur d'émission, qui se retrouvera dans le champ "*KeyID*" du paquet émis, et l'identificateur de réception, qui sert pour le paquet reçu.
— Un identifiant de connexion TCP, {adresse IP source, port source, adresse IP de destination, port destination}. Certains des éléments de ce tuple peuvent être spécifiées par un intervalle (par exemple, « ports de 8080 à 8099 ») ou même par un joker (*).
— La clé elle-même.
— L'algorithme de hachage utilisé (voir section 7.1), par exemple SHA-1.
— Le choix de traitement des autres options TCP (les met-on dans le MAC ou pas ?).

Le MKT doit rester stable pendant toute une connexion TCP.

À partir du MKT va être calculée la clé de session (section 5.2), en incluant des paramètres supplémentaires comme l'ISN ("*Initial Sequence Number*", voir la section 3.3 du RFC 793), en utilisant une fonction cryptographique nommé KDF ("*Key Derivation Function*").

L'un des plus nets avantages de TCP-AO sur l'ancienne méthode du RFC 2385 est la possibilité de changer d'algorithme de hachage. La section 7 décrit les mécanismes généraux, les algorithmes actuellement utilisables étant dans un autre document, le RFC 5926 (en effet, comme l'ont montré les attaques contre MD5 et SHA-1, les algorithmes de cryptographie doivent être changés plus souvent que les protocoles, ce qui justifie un document séparé, qui rendra plus simple ce changement). Un autre registre IANA existe, pour ces algorithmes <https://www.iana.org/assignments/tcp-parameters/tcp-parameters.xhtml#tcp-parameters-3>. La section 7.1 rappelle le fonctionnement d'un MAC : en échange d'une suite de bits et d'une clé, la fonction de hachage produit un résumé. Si la fonction de hachage est cryptographiquement bien choisi, on ne peut pas (autrement que par force brute) fabriquer une suite de bits qui donne le même résumé. L'ajout de la clé sert à assurer en outre que la suite de bits vient bien de l'émetteur attendu, seul, avec le récepteur, à connaître la clé.

La même section décrit précisement les parties du paquet qui doivent être utilisées comme point d'entrée pour la fonction de calcul du MAC. On y trouve le « pseudo-en-tête » IP comme dans la section 3.1 du RFC 793 mais aussi l'en-tête TCP (avec ou sans les options, comme indiqué dans la MKT) et les données.

Suivant le même modèle, la section 7.2 décrit le fonctionnement de la KDF, qui permet de calculer la clé de session à partir de la MKT. Un des paramètres d'entrée est l'état de la connexion TCP (incluant par exemple les numéros de séquence TCP initiaux), faisant ainsi des clés de session uniques par connexion TCP.

Rappelons que TCP-AO ne fournit pas une solution complète de gestion de clés. Il ne permet pas d'en changer de manière synchronisée en cours de session (contrairement à TLS), il ne permet pas de les négocier au début de la session... Tout doit se faire "*out-of-band*". La section 7.3, consacrée aux clés, recommande juste de suivre les bonnes pratiques habituelles, comme déjà recommandé par le RFC 3562. Les utilisateurs doivent donc veiller par eux-mêmes à ce que les clés aient une longueur suffisante, doivent éviter de partager les MKT entre connexions indépendantes (par exemple avec des pairs BGP différents), etc.

TCP-AO fournit toutefois quelques mécanismes pour jouer avec les clés, comme l'indication de l'identificateur de la MKT, qui permet de coordonner le passage d'une MKT à une autre (section 8.1). Le choix des MKT et l'attribution des identificateurs se fait en dehors du protocole mais le démarrage de l'utilisation effective peut être synchronisé par le champ "*KeyID*" de TCP-AO.

D'autres mécanismes de sécurité sont décrits en section 8, comme les solutions anti-rejeu de la section 8.2. Comme les numéros de séquence TCP ne sont stockés que sur 32 bits, et qu'ils peuvent donc

légitimement être réutilisés pendant une connexion TCP de longue durée, TCP-AO ajoute une extension à ces numéros, la SNE ("*Sequence Number Extension*") qui, combinée avec le numéro de séquence, donne effectivement un numéro unique à chaque octet transmis. (Ou quasi-unique puisqu'il ne sera réutilisé qu'au bout de 100 exa-octets transmis. Et rappelez-vous que la MAC inclus les données donc une collision accidentelle ne se produira que si le numéro de séquence **et les données** sont identiques. En pratique, on peut dormir tranquille.)

La meilleure cryptographie du monde ne sert que s'il y a une liaison solide avec le canal qu'elle protège. La section 9 discute donc des interactions entre TCP et son option d'authentification TCP-AO. D'abord, l'interface de programmation (section 9.1). Aux commandes classiques de TCP (`OPEN`, `SEND`, etc, ou, pour utiliser les termes de l'API "*socket*", `connect()`, `write()`, etc) viennent s'ajouter des commandes pour configurer les MKT, et pour choisir la MKT active. J'avoue ne pas savoir quelle forme elles prendront exactement sur Unix.

Ensuite, TCP lui-même doit être modifié pour, si une MKT est active, signer les données et inclure cette signature dans le champ Options du segment. Il faut aussi évidemment la vérifier à la réception (sections 9.4 et 9.5). Cet ajout dans un champ Options dont la taille est très limitée (40 octets en tout) ne va pas sans mal. La section 9.6 discute de la taille de l'en-tête et explique que, si on utilise TCP-AO, on ne peut pas espérer utiliser en même temps **toutes** les autres options TCP normalisées.

Aucune solution de sécurité n'est parfaite et toutes apportent leurs propres problèmes. La réduction de la place libre pour les options est un bon exemple. Un autre est le cas d'une machine qui redémarre alors qu'une connexion TCP était ouverte. Sans TCP-AO, elle enverra des paquets RST ("*ReSeT*") lors de la réception des paquets TCP de son pair, puisque ces paquets ne correspondront à aucune connexion connue, le souvenir s'étant perdu lors du redémarrage. Cela permettra d'informer rapidement le pair que la connexion TCP doit être considérée comme terminée. Avec TCP-AO, ce n'est plus possible : la machine ayant tout oublié, y compris les clés de session, elle ne peut plus construire des segments TCP authentifiés. Ses RST seront donc ignorés par le pair, qui devra donc attendre l'expiration d'un délai de garde pour comprendre que la connexion TCP est finie (section 9.7 du RFC). C'est dommage mais c'est fait exprès : empêcher les RST « pirates » était après tout une des principales motivations pour l'authentification TCP. Notre RFC recommande donc d'utiliser les "*keepalives*" du RFC 1122 ou bien ceux fournis par les applications utilisés ("*keepalives*" BGP du RFC 4271 ou bien les options `ClientAlive*` ou `ServerAlive*` de OpenSSH). Ces solutions ne sont pas parfaites (pour BGP, il est recommandé d'ajouter le démarrage en douceur du RFC 4724.) Une autre option est d'enregistrer les clés de session sur disque.

Comme vu plus haut, TCP-AO ne protège que TCP, pas les éventuels paquets ICMP qui se glisseraient dans la communication. La recommandation de notre RFC (section 9.8) est donc d'accepter avec prudence et conservatisme les paquets ICMP. Par exemple, les paquets ICMP ayant le type 3 ("*Destination unreachable*" et les codes 2 à 4 ("*protocol unreachable*", "*port unreachable*", ...) devraient être ignorés, car ils peuvent casser une connexion TCP-AO, sans avoir eux-même besoin de s'authentifier. La même section demande que ce comportement soit configurable. Cette recommandation est analogue à celle qui s'applique à IPsec (section 6.1.1 du RFC 4301).

Normalement, la spécification du protocole devrait s'arrêter là. Mais, dans le monde réel, il faut aussi tenir compte des engins intermédiaires ("*middleboxes*", RFC 3234) qui examinent les paquets et se permettent de les rejeter ou de les modifier s'ils ne correspondent pas à leurs préjugés. La section 11 couvre les interactions avec ces engins (certains routeurs, les pare-feux, etc). Si la "*middlebox*" ne modifie pas les adresses IP, TCP-AO devrait passer sans problème. Si elle change certaines options TCP, il faut configurer TCP-AO pour ignorer les options dans le calcul de la MAC, ce qui affaiblit la sécurité (puisqu'un attaquant pourrait en faire autant, par exemple en supprimant le "*window scaling*" ce qui empêcherait le fonctionnement normal de TCP).

Mais si la "*middlebox*" change les adresses IP, ce qui est le cas des routeurs NAT (section 11.2), TCP-AO ne peut plus fonctionner du tout. Les seules solutions sont d'encapsuler le flux TCP ou de compter sur l'extension NAT du RFC 6978.

Et les implémentations ? À l'heure actuelle, rien dans Linux (il y a un projet <http://lwn.net/Articles/290963/>), FreeBSD ou NetBSD. Et je ne connaissais pas non plus de mise en œuvre pour les routeurs Cisco ou Juniper, sans doute en partie en raison des innombrables brevets qui infestent le secteur informatique. Fin 2011, lorsque la question avait été étudiée à l'IETF pour un autre protocole, la conclusion avait été qu'AO restait toujours uniquement sur le papier, hélas. Depuis, il semble (2017) que Juniper ait une mise en œuvre d'AO. Vous trouverez davantage d'informations concrètes sur ce dépôt GitHub <https://github.com/TCP-AO/>.